A Project Report

on

FITWORLD: Your Fitness Buddy

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Ms. AANCHAL VIJ:
ASSISTANT PROFESSOR

Submitted By

SARTHAK AGARWAL– 19SCSE1180005
ROHIT BASRA– 19SCSE1180123

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"*FITWORLD: Your Fitness Buddy*"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the **School of Computing Science and Engineering** of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY, 2021 TO DECEMBER, 2021**, under the supervision of **MS. AANCHAL VIJ, Assistant Professor, Department of Computer Science and Engineering** of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

Sarthak Agarwal- 19SCSE1180005

Rohit Basra- 19SCSE1180123

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Ms. Aanchal Vij

Assistant Professor

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **SARTHAK AGARWAL: 19SCSE1180005 AND ROHIT BASRA: 19SCSE1180123** has been held on __________________ and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021

Place: Greater Noida

ABSTRACT

In the modern era of work from home and the recent Covid=19 times , Fitness or we can say obesity has become a major problem that we face. The sudden increase in use of technology in our life has embedded itself.

For the emergence of those people we are creating a fitness app namely **FITWORLD** that helps people to achieve their respective goals by providing them workout schedules and diet plans according to their respective goals. We have studied fitness patterns of different people with different goals and with different BMI and used that study for our recommendation.These recommendations are easy to follow and also increase immunity which further protects from Covid.

The various tools and technology that we are using here are:

- Android Studio
- Kotlin
- XML
- Draw.io
- Figma

As a result we are attempting to make an app namely **Fitworld** using above mentioned tools and technology.

As a future scope we are aiming to make our country Fit and Healthy by helping people maintain a healthy lifestyle by using our app.

List of Tables

Student Details:

S.No	Name	Enrollment No.	Admission No.	Program /Branch	Sem	Mobile	Email
1.	Sarthak Agarwal	19021180004	19SCSE1180005	Btech CSE – AIML	5	9811402220	agarwal.sarthak262012@gmail.com
2.	Rohit Basra	19021011638	19SCSE1180123	Btech CSE - AIML	5	8826201279	rohitbasra76@gmail.com

Faculty Details:

Name	Designation	Email id	Mobile No.
Ms. Aanchal Vij	Assistant Professor	aanchal.vij@galgotiasuniversity.edu.in	8146325511

List of Figures

Acronyms:

B.Tech	Bachelor of Technology
M.Tech	Master of Technology
BCA	Bachelor of Computer Applications
MCA	Master of Computer Applications
B.Sc. (CS)	Bachelor of Science in Computer Science
M.Sc. (CS)	Master of Science in Computer Science
SCSE	School of Computing Science and Engineering

Table of Contents

Introduction

Being unhealthy and getting obese is a major problem that we all are facing due to our uncertain schedule and work from home caused due to covid which led to effect our lifestyle and our health also. This led to cause some serious problem like cholesterol increment, heart issue , rise of uric acid and lack of physical activity makes our body stiff which leads to back pain.We ourselves have faced the problem and trying to help our society with an app which provides people with exclusive workout plans according to their body structure and their goals which basically will help to maintain a healthy lifestyle and keep our body fit.

We have firstly studied various body types and according to their BMI have designed diet and workout plans and have tested these plans on different individuals which works successfully based on these studies we are recommending these plans.

Since, mental health is equally important to physical health, the application also helps in reducing mental fatigue by creating a habit of regular mental health exercises so that the users can be relieved of their stress.

The application that we are designing, namely “FITWORLD” requires the following tools and technologies: Android Studio, IntelliJ Idea, Kotlin, Ktor, REST APIs, XML, draw.io, Figma.

Formulation Of Problem

We specifically chose to work on this project because we experienced this problem first hand. During the pandemic when everyone had to work from home, people took to their comforts and forgot about their physical and mental health. While the traumatic effects of COVID-19 deteriorated people's mental health, many became couch potatoes and gained weight.

We were no different from this. We also experienced severe mental stress and faced physical challenges. The pandemic made it difficult for all of us to lead a healthy lifestyle. This unhealthy lifestyle often leads to lack of efficiency and productivity. Many surveys indicate that people wanted to get out of this rut but weren't able to sustain their actions of self development.

So, when we were presented with the opportunity of doing a project this semester, we just jumped upon this idea and started working towards creating value for everyone in this world.

Our aim is to create a product that adds value to user's daily life by helping them reach their fitness goals. The application helps in creating a habit of physical and mental fitness exercises by providing tailor-made plans in accordance with the user's profile to boost their personal development. This product takes away all the worries of the users and helps them grow and perform at optimum levels all the time.

Tool and Technology Used

The various tools and technology that we used in our app are:

- Android studio
- IntelliJ Idea
- Kotlin
- Ktor
- XML
- draw.io
- Figma

Literature Survey

The three fitness apps selected for this study were based on the Functional Triad as well as from a focus group with fitness professionals to ensure each app fit strongly with one of the three types of technology in the Functional Triad. The Functional Triad suggests that there are three functions of technology in the way people react or use them: (I) a medium provides an experience; (II) a tool increases capability; and (III) a social actor creates relationships. Each of the three apps in the present study had a primary function as a medium that provided the user an experience in terms of a dashboard coordinating fitness efforts, a tool that increased capability in terms of prefabricated workouts, or social actor that created relationships with others seeking to improve fitness via a social media platform. Apps were available for free on Android and iOS.

A one-group pre-posttest design was utilized since a goal of this study was to examine technology usage attrition in addition to effectiveness of apps. Participants were allowed to use all three apps simultaneously to determine which app functions, as outlined in the Functional Triad (medium, tool, social actor), were utilized and found to be effective. After receiving IRB approval, 64 participants (17 men, 47 women) aged 18 or older were recruited in-person from a Midwest suburban fitness center between June 2014 and January 2015. An already active population was selected for the study because effects of fitness apps were being tested against exercise without technology. At recruitment, participants downloaded the three apps, but were not told the purpose of each app as a medium, tool, and social actor. Participants were told to use the apps however they preferred, as if they had found and downloaded them on their own.

Participants completed a validated TPB and exercise survey at pretest and posttest regarding attitude, subjective norm, perceived behavioral control, and behavioral intention over exercise and exercise with apps. The bipolar adjective scales measured exercise attitude by descriptive paired adjective categories including useless/useful, foolish/wise, harmful/beneficial, unenjoyable/enjoyable, unpleasant/pleasant, boring/interesting, and stressful/relaxing. Subjective norm measured peer support and peer approval of exercise and exercise with apps. Perceived behavioral control examined control over and barriers to exercise and exercise with apps. According to the authors of the instrument , a subscale total score is calculated by summing scores of individual items (each ranging from 1 to 7) in each construct. Lastly, behavioral intention to exercise and to exercise with apps asked about the number of times the participant intended to exercise and exercise with apps over the next 2 weeks in six groups (0, 1–3, 4–6, 7–9, 10–12, and 12 times or more).

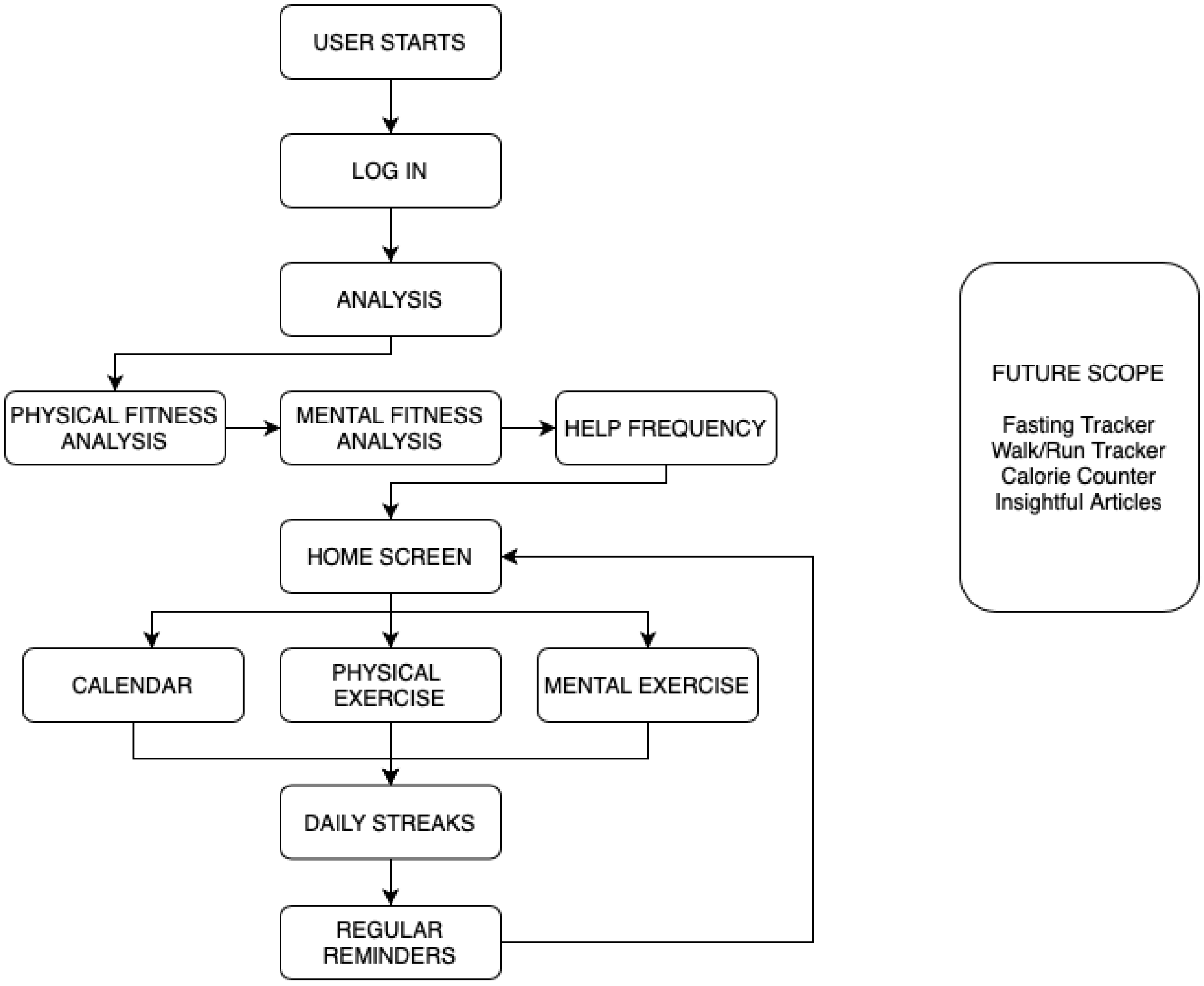
Participant app usage was tracked at months 1, 3, and 5 after download to determine usage and perceived fitness. Participants received an email with a checkpoint survey that inquired about times per month of app usage for each app and minutes of usage per each time for each app. The amount of app use was calculated by multiplying the frequency of app use per month and the minutes of use each time. At month 5, participants also received the posttest survey via email. Additionally, fitness perception was measured at pretest, checkpoint one, checkpoint two, and checkpoint three (posttest). Measurements of perceived fitness on a 12-point scale included cardiovascular fitness, muscular strength, muscular endurance, flexibility, and body composition. If participants did not respond to the initial checkpoint email, they were sent one reminder email.

Survey data were analyzed using statistical software Stata . Our analysis focused on comparisons of pre-post differences and comparisons of those who used apps (users) and those who did not (non-users). Paired sample *t*-tests analyzed data for pretest to posttest comparisons of (I) individual item scores for constructs of attitude, subjective norm, and perceived behavioral control; (II) subscale total scores for attitude, subjective norm, and perceived behavioral control for each exercise and exercise with apps; and (III) perceived fitness scores. A pre-post difference in behavioral intention over exercise and exercise with apps was examined using a sign test for matched pairs. A sign test is appropriate for ordinal variables because the non-parametric test compares scores of matched or paired samples without requiring the outcome's distribution to be normal or symmetric . The sign test produces the results of three hypothesis tests: whether the median of the differences between paired observations is positive, negative, and zero .

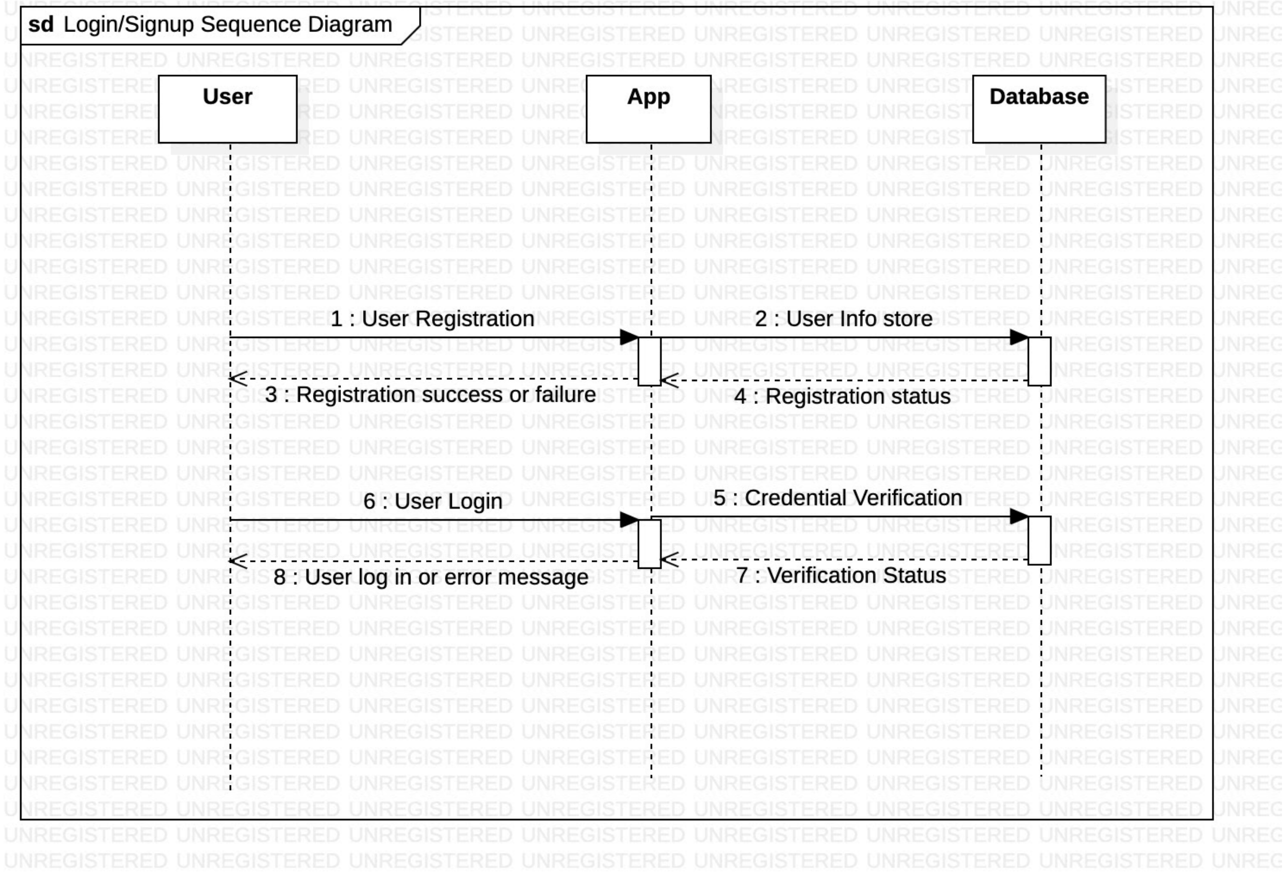
Baseline characteristics of users and non-users were examined to identify types or statuses of individuals who are more likely to use mobile apps for exercise. Fisher's exact tests were used for categorical (e.g., gender, race, marital status) and ordinal (e.g., behavioral intention) independent variables, and simple logistic regression was used for continuous independent variables (e.g., subscale total scores of TPB constructs) with app use status as the dependent variable. In addition, we examined whether app use is associated with TPB constructs at posttest. Independent sample *t*-tests were used to examine differences between users and non-users in subscale total scores for attitude, subjective norm, and perceived behavioral control at posttest. Fisher's exact tests were used to compare behavioral intention over exercise and exercise with apps at posttest between users and non-users. *T*-tests were used to examine whether the amount of app use (minutes of use per month) was different between two different checkpoints for each app. Simple linear regression was used to examine whether the amount of app use (minutes of use per month) is associated with TPB constructs or perceived fitness at posttest.

App usage and effectiveness appears to have a connection to usefulness (attitude) and to perceived difficulties of exercising using apps (perceived behavioral control). Exercise and exercise using apps are not influenced by peer influence (subjective norm). Intention to exercise using these particular apps decreased (behavioral intention). Those who utilized the apps were more likely to have a positive attitude about the apps. Usefulness and perceived difficulties in particular should be considered with future app development. App usefulness and ease of use may be facilitated by using health behavior theories to guide development.

Project Design



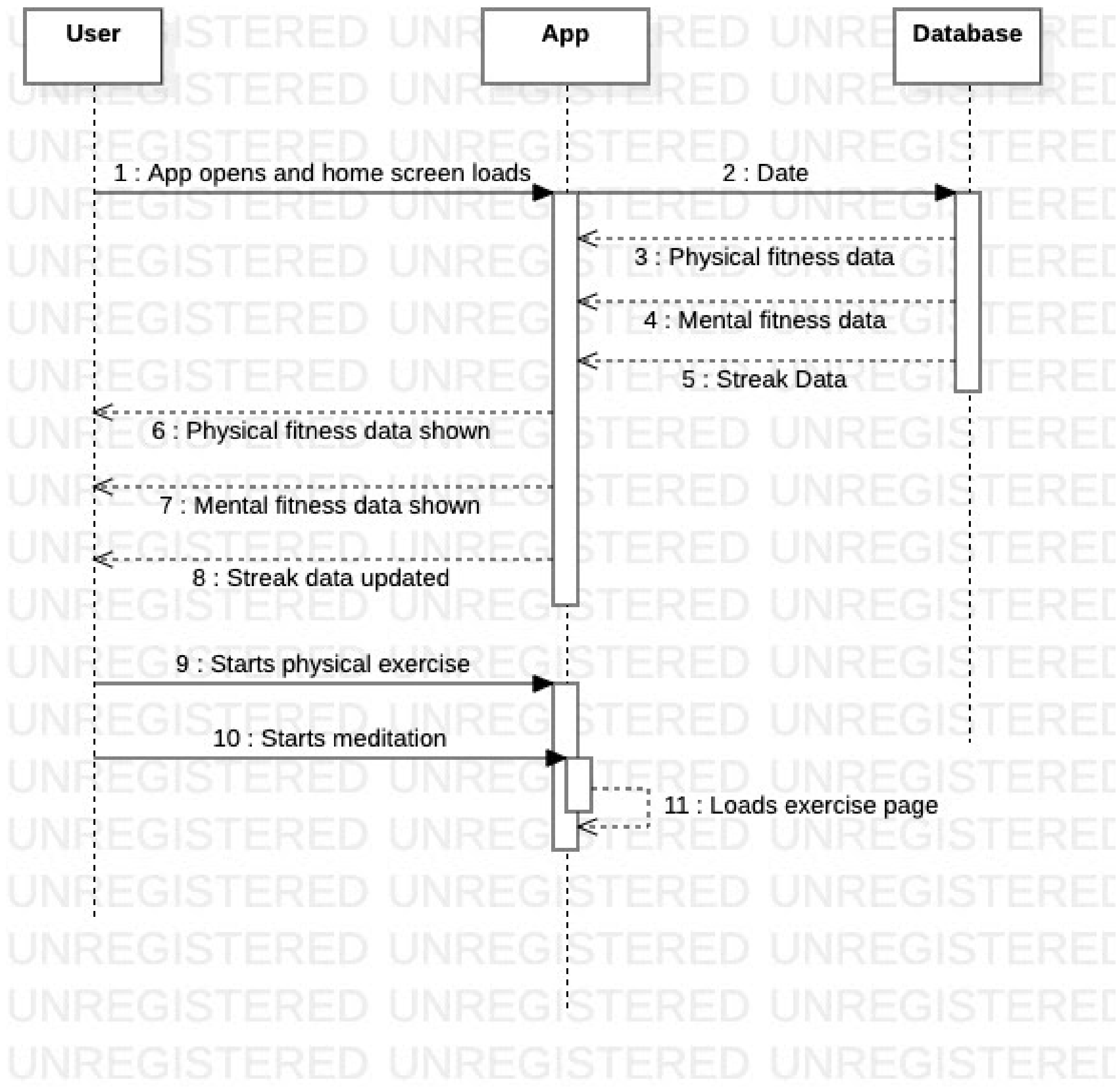
- **Login Page Layout**



sd Login/Signup Sequence Diagram
User
App
Database
1 : User Registration
2 : User Info store
3 : Registration success or failure
4 : Registration status
6 : User Login
5 : Credential Verification
8 : User log in or error message
7 : Verification Status

In the Login Page it asks users for registration to continue using the app for which users have to add his/her details to register themselves.After successful verification of their credential details then users will be allowed to login using their username and password. If the login details are correct then It will be directed to Homepage else it will show invalid login.

- **HomePage**


User
App
Database
1 : App opens and home screen loads
2 : Date
3 : Physical fitness data
4 : Mental fitness data
5 : Streak Data
6 : Physical fitness data shown
7 : Mental fitness data shown
8 : Streak data updated
9 : Starts physical exercise
10 : Starts meditation
11 : Loads exercise page

Homepage includes feature of :

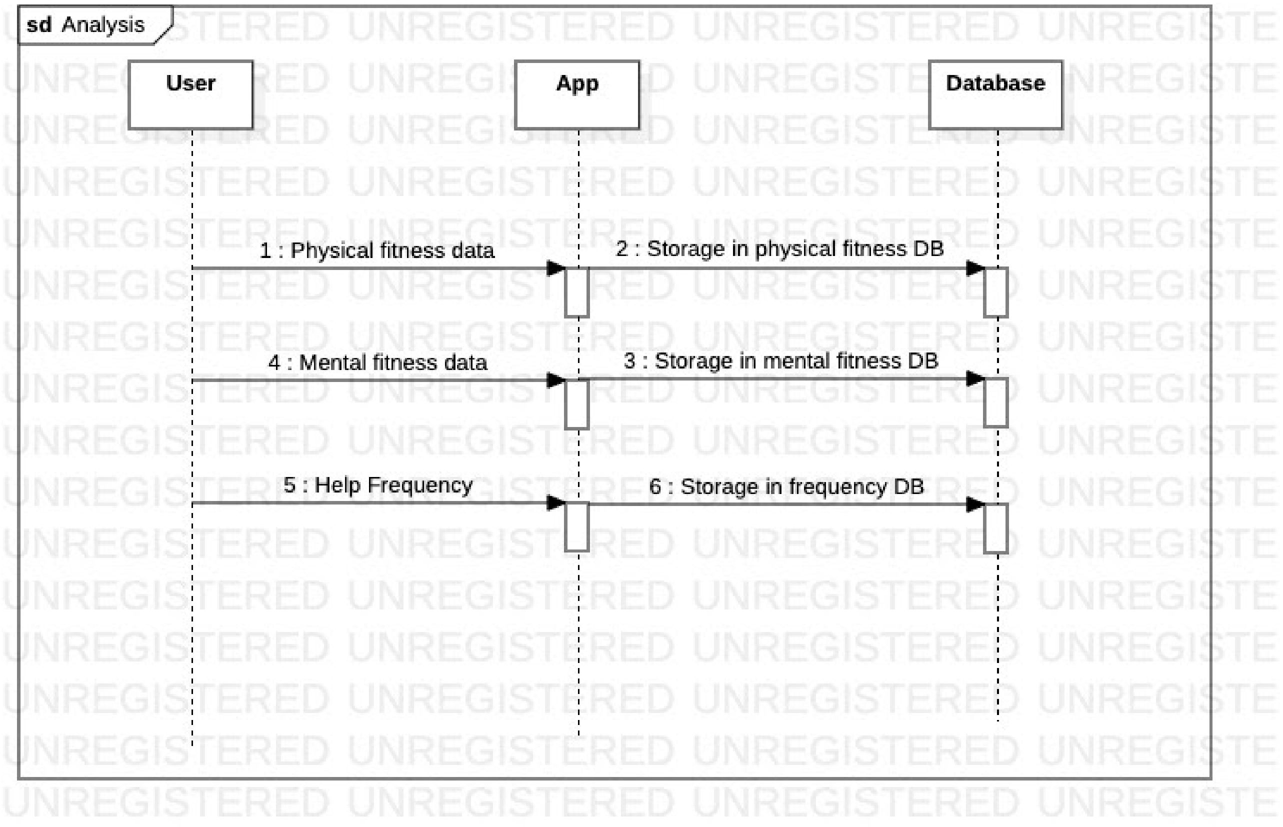
- Physical Fitness Data:
    - It stores workout information about the workouts that the user has to perform.
    - It also stores the data after physical exercise has been performed like how much weight used for how many reps and for how long exercise were done
- Mental Fitness Data:
    - It stores meditation information like meditation name,

audio track and other metadata for the meditation that the user has to perform.

  - It also keeps track of your stress level and recommend various relief measures

- Streak Data: It shows for how long you are being regular in exercising and provide you rewards for achieving streak
- Physical Exercise: It recommend you exercise according to your schedule and level
- Meditation: It recommend you to meditate according to your stress level

- **Analysis**


sd Analysis
User
App
Database
1 : Physical fitness data
2 : Storage in physical fitness DB
4 : Mental fitness data
3 : Storage in mental fitness DB
5 : Help Frequency
6 : Storage in frequency DB
UNREGISTERED

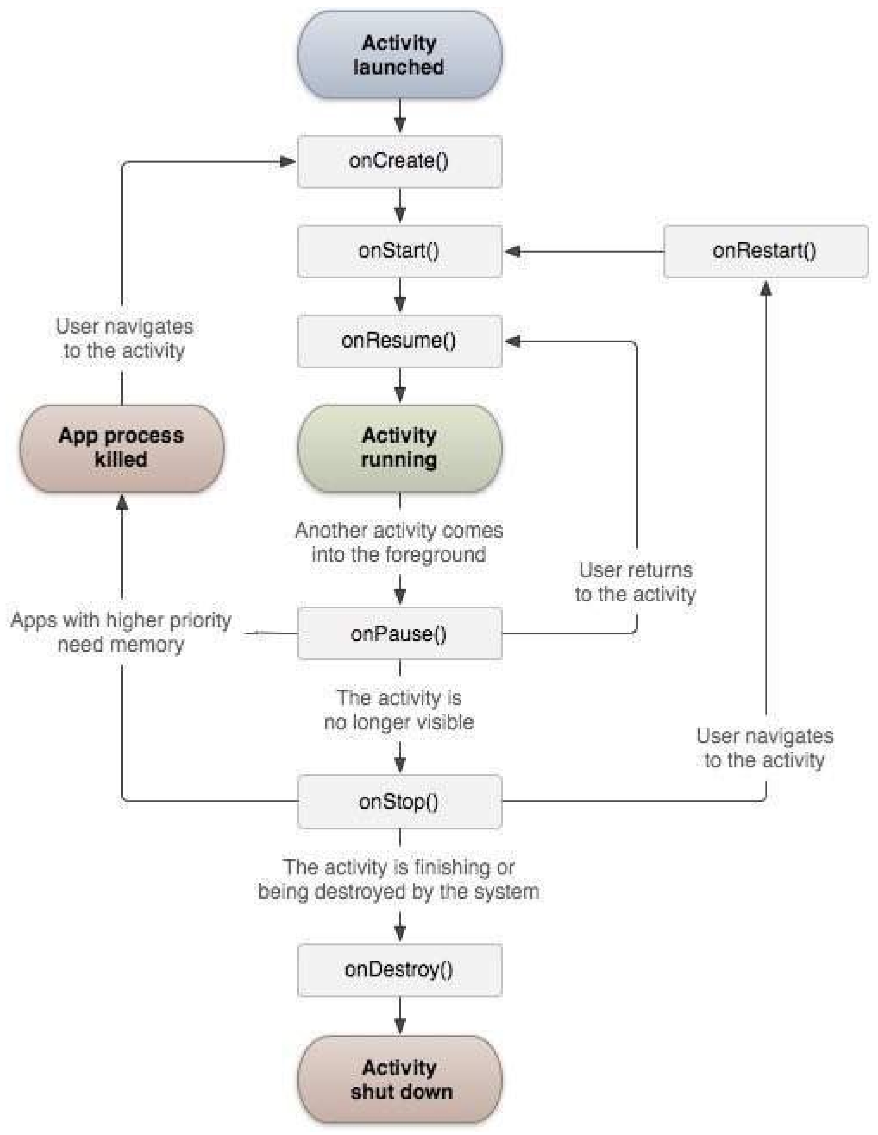
It stores the Physical and Mental fitness data of the user in the database.

Analysis is done on this Physical Fitness Data and Mental fitness data on the basis of which, physical and mental exercises would be provided to the user. Data of help frequency is also recorded which is used to determine the number of times a week the user would do the exercises. The system would only prompt the user to do the exercises on these pre-decided days of the week.

Implementation

● Activity Lifecycle

The Android system initiates its program within an Activity starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram:


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Sr.No	Callback & Description
1	**onCreate()** This is the first callback and called when the activity is first created.
2	**onStart()** This callback is called when the activity becomes visible to the user.
3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and is called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()**

	This callback is called when the activity is no longer visible.
6	**onDestroy()** This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

Example Implementation

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
   String msg = "Android : ";

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(msg, "The onCreate() event");
   }

   /** Called when the activity is about to become visible. */
   @Override
   protected void onStart() {
      super.onStart();
      Log.d(msg, "The onStart() event");
   }

   /** Called when the activity has become visible. */
   @Override
   protected void onResume() {
      super.onResume();
      Log.d(msg, "The onResume() event");
   }

   /** Called when another activity is taking focus. */
   @Override
   protected void onPause() {
      super.onPause();
      Log.d(msg, "The onPause() event");
   }

   /** Called when the activity is no longer visible. */
   @Override
   protected void onStop() {
      super.onStop();
      Log.d(msg, "The onStop() event");
   }

   /** Called just before the activity is destroyed. */
   @Override
   public void onDestroy() {
      super.onDestroy();
      Log.d(msg, "The onDestroy() event");
   }
}
```

- **REST APIs**

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding.

An API is a set of definitions and protocols for building and integrating application software. It’s sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response). For example, the API design for a weather service could specify that the user supply a zip code and that the producer reply with a 2-part answer, the first being the high temperature, and the second being the low.

In other words, if you want to interact with a computer or system to retrieve information or perform a function, an API helps you communicate what you want to that system so it can understand and fulfill the request.

You can think of an API as a mediator between the users or clients and the resources or web services they want to get. It's also a way for an organization to share resources and information while maintaining security, control, and authentication—determining who gets access to what.

Another advantage of an API is that you don't have to know the specifics of caching—how your resource is retrieved or where it comes from.

REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it's language-agnostic, as well as readable by both humans and machines.

Something else to keep in mind: Headers and parameters are also important in the HTTP methods of a RESTful API HTTP request, as they contain important identifier information as to the request's metadata, authorization, uniform resource identifier (URI), caching, cookies, and more. There are request headers and response headers, each with their own HTTP connection information and status codes.

In order for an API to be considered RESTful, it has to conform to these criteria:

- A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
- Stateless client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.
- Cacheable data that streamlines client-server interactions.
- A uniform interface between components so that information is transferred in a standard form. This requires that:
    - resources requested are identifiable and separate from the representations sent to the client.
    - resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.
    - self-descriptive messages returned to the client have enough information to describe how the client should process it.
    - hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.
- A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.
- Code-on-demand (optional): the ability to send executable code from the server to the client when requested, extending client functionality.

Though the REST API has these criteria to conform to, it is still considered easier to use than a prescribed protocol like SOAP (Simple Object Access Protocol), which

has specific requirements like XML messaging, and built-in security and transaction compliance that make it slower and heavier.
In contrast, REST is a set of guidelines that can be implemented as needed, making REST APIs faster and more lightweight, with increased scalability—perfect for Internet of Things (IoT) and mobile app development.

- **Public APIs** are publicly available to developers and other users with minimal restriction. They may require registration, use of an API Key or OAuth, or maybe completely open. They focus on external users, to access data or services.
- **Partner APIs** are APIs exposed by/to the strategic business partners. They are not available publicly and need specific entitlement to access them. Like open APIs, partner APIs are the tip of the iceberg because they are the most visible ones and are used to communicate beyond the boundaries of the company. They are usually exposed to a public API developer portal that developers can access in self-service mode. While open APIs are completely open, there is an onboarding process with a specific validation workflow to get access to partner APIs.
- **Internal APIs,** aka **private APIs,** are hidden from external users and only exposed by internal systems. Internal APIs are not meant for consumption outside of the company but rather for use across different internal development teams for better productivity and reuse of services. A good governance process comprises exposing them to an internal API developer portal that connects to the internal IAM systems to authenticate and allow users to access the right set of APIs.
- **Composite APIs** combine multiple data or service APIs. They are built

using the API orchestration capabilities of an API creation tool. They allow developers to access several endpoints in one call. Composite APIs are useful, for example, in a microservices architecture pattern where you need information from several services to perform a single task.

To leverage these different types of APIs, we must follow certain protocols. A protocol provides defined rules for API calls. It specifies the accepted data types and commands. Let’s look at the major types of protocols for APIs:

1. REST

REST (short for Representational State Transfer) is a web services API. REST APIs are a key part of modern web applications, including Netflix, Uber, Amazon, and many others. For an API to be RESTful, it must adhere to the following rules:

- Stateless—A REST API is stateless in nature, Client-Server Architecture
- Uniform Interface—A client and server should communicate with one another via HTTP (HyperText Transfer Protocol) using URIs (Unique Resource Identifiers), CRUD (Create, Read, Update, Delete), and JSON (JavaScript Object Notation) conventions.
- Client-Server—The client and server should be independent of each other. The changes you make on the server shouldn’t affect the client and vice versa.
- Cache—The client should cache the responses as this improves the user experience by making them faster and more efficient.
- Layered—The API should support a layered architecture, with each layer contributing to a clear hierarchy. Each layer should be loosely coupled and allow for encapsulation.

2. SOAP

SOAP (simple object access protocol) is a well-established protocol similar to REST in that it’s a type of Web API.
SOAP has been leveraged since the late 1990s. SOAP was the first to standardize the way applications should use network connections to manage services.
But SOAP came with strict rules, rigid standards were too heavy, and, in some situations, very resource-intensive. Except for existing on-premise scenarios, most developers now prefer developing in REST over SOAP.

3. RPCAn RPC is a remote procedure call protocol. They are the oldest and simplest types of APIs. The goal of an RPC was for the client to execute code on a server. XML-RPC used XML to encode its calls, while JSON-RPC used JSON for the encoding.

Both are simple protocols. Though similar to REST, there are a few key differences. RPC APIs are very tightly coupled, so this makes it difficult to maintain or update them.
To make any changes, a new developer would have to go through various RPCs documentation to understand how one change could affect the other.
APIs play a key role in the development of any application. And REST has become the preferred standard for building applications that communicate over the network. REST fully leverages all the standards that power the World Wide Web and is simpler than traditional SOAP-based web services. Unlike RPC, it allows for a loosely coupled layered architecture to maintain easily or update them.

- **<u>Retrofit</u>**

Retrofit is a type-safe REST client for Android and Java which aims to make it easier to consume RESTful web services. We'll not go into the details of Retrofit 1.x versions and jump onto Retrofit 2 directly which has a lot of new features and a changed internal API compared to the previous versions.

Interceptors are a powerful mechanism present in OkHttp that can monitor, rewrite, and retry calls.Interceptors can be majorly divided into two categories:

1. **Application Interceptors** : To register an application interceptor, we need to call addInterceptor() on OkHttpClient.Builder
2. **Network Interceptors** : To register a Network Interceptor, invoke addNetworkInterceptor()instead of addInterceptor()

```
class APIClient { private static Retrofit retrofit =
null; static Retrofit getClient() {
HttpLoggingInterceptor interceptor = new
HttpLoggingInterceptor();
interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
OkHttpClient client = new
OkHttpClient.Builder().addInterceptor(interceptor).build
(); retrofit = new Retrofit.Builder()
.baseUrl("https://reqres.in")
.addConverterFactory(GsonConverterFactory.create())
.client(client) .build(); return retrofit; } }
```

The getClient() method in the above code will be called every time while setting up a Retrofit interface. Retrofit provides a list of annotations for each of the HTTP methods: @GET, @POST, @PUT, @DELETE, @PATCH or @HEAD.

```
interface APIInterface {

    @GET("/api/unknown")
    Call<MultipleResource> doGetListResources();

    @POST("/api/users")
    Call<User> createUser(@Body User user);

    @GET("/api/users?")
    Call<UserList> doGetUserList(@Query("page") String
page);

    @FormUrlEncoded
    @POST("/api/users?")
    Call<UserList> doCreateUserWithField(@Field("name")
String name, @Field("job") String job);
}
```

```
public class MultipleResource { @SerializedName("page")
public Integer page;
@SerializedName("per_page") public Integer perPage;
@SerializedName("total") public Integer total;
@SerializedName("total_pages") public Integer totalPages;
@SerializedName("data") public List<Datum> data = null;
public class Datum {
@SerializedName("id") public Integer id;
@SerializedName("name") public String name;
@SerializedName("year") public Integer year;
@SerializedName("pantone_value") public StringpantoneValue;
} }
```

```
public class MultipleResource {
@SerializedName("page") public Integer page;
@SerializedName("per_page") public Integer perPage;
@SerializedName("total") public Integer total;
@SerializedName("total_pages") public Integer totalPages;
@SerializedName("data") public List<Datum> data = null;
public class Datum {
@SerializedName("id") public Integer id;
@SerializedName("name") public String name;
@SerializedName("year") public Integer year;
@SerializedName("pantone_value") public String pantoneValue;
} }
```

- **Storage in Android**

Android uses a file system that's similar to disk-based file systems on other platforms. The system provides several options for you to save your app data:

- App-specific storage: Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
- Shared storage: Store files that your app intends to share with other apps, including media, documents, and other files.
- Preferences: Store private, primitive data in key-value pairs.
- Databases: Store structured data in a private database using the Room

persistence library.

	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App-specific files	Files meant for your app's use only	From internal storage, getFilesDir() or getCacheDir() From external storage,getExternalFilesDir() orgetExternalCacheDir()	Never needed for internal storage Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher	No	Yes

Media	Shareable media files (images, audio files, videos)	MediaStore API	READ_EXTERNAL_STORAGE when accessing other apps' files on Android 11 (API level 30) or higher READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGEwhen accessing other apps' files on Android 10 (API level 29) Permissions are required for all files on Android 9 (API level 28) or lower	Yes, though the other app needs the READ_EXTERNAL_STORAGEpermission	No

Documents and other files	Other types of shareable content, including downloaded files	Storage Access Framework	None	Yes, through the system file picker	No
App preferences	Key-value pairs	Jetpack Preferences library	None	No	Yes
Database	Structured data	Room persistence library	None	No	Yes

The solution you choose depends on your specific needs:

How much space does your data require?

Internal storage has limited space for app-specific data. Use other types of storage if you need to save a substantial amount of data.

How reliable does data access need to be?

If your app's basic functionality requires certain data, such as when your app is starting up, place the data within internal storage directory or a database.

App-specific files that are stored in external storage aren't always accessible because some devices allow users to remove a physical device that corresponds to external storage.

What kind of data do you need to store?

If you have data that's only meaningful for your app, use app-specific storage. For shareable media content, use shared storage so that other apps can access the content. For structured data, use either preferences (for key-value data) or a database (for data that contains more than 2 columns).

Should the data be private to your app?

When storing sensitive data—data that shouldn't be accessible from any other app—use internal storage, preferences, or a database. Internal storage has the added benefit of the data being hidden from users.

Android provides two types of physical storage locations: *internal storage* and *external storage*. On most devices, internal storage is smaller than external storage. However, internal storage is always available on all devices, making it a more reliable place to put data on which your app depends.

Removable volumes, such as an SD card, appear in the file system as part of external storage. Android represents these devices using a path, such as /sdcard.

Android defines the following storage-related permissions: READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, andMANAGE_EXTERNAL_STORAGE.

On earlier versions of Android, apps needed to declare the

READ_EXTERNAL_STORAGE permission to access any file outside the app-specific directories on external storage. Also, apps needed to declare the WRITE_EXTERNAL_STORAGEpermission to write to any file outside the app-specific directory.

More recent versions of Android rely more on a file's purpose than its location for determining an app's ability to access, and write to, a given file. In particular, if your app targets Android 11 (API level 30) or higher, theWRITE_EXTERNAL_STORAGE permission doesn't have any effect on your app's access to storage. This purpose-based storage model improves user privacy because apps are given access only to the areas of the device's file system that they actually use.

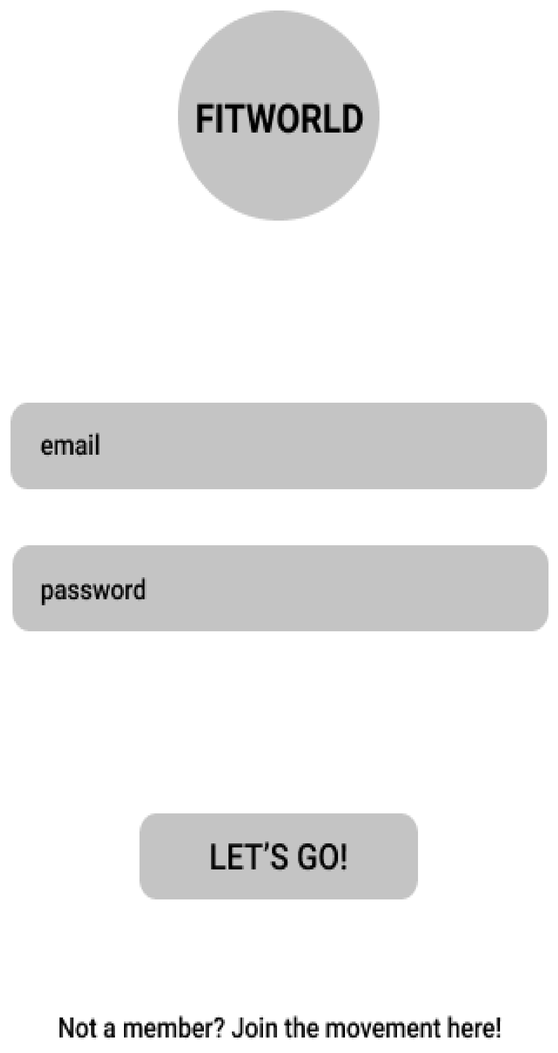
Android 11 introduces the MANAGE_EXTERNAL_STORAGE permission, which provides write access to files outside the app-specific directory and MediaStore. To learn more about this permission, and why most apps don't need to declare it to fulfill their use cases, see the guide on how to manage all files on a storage device.

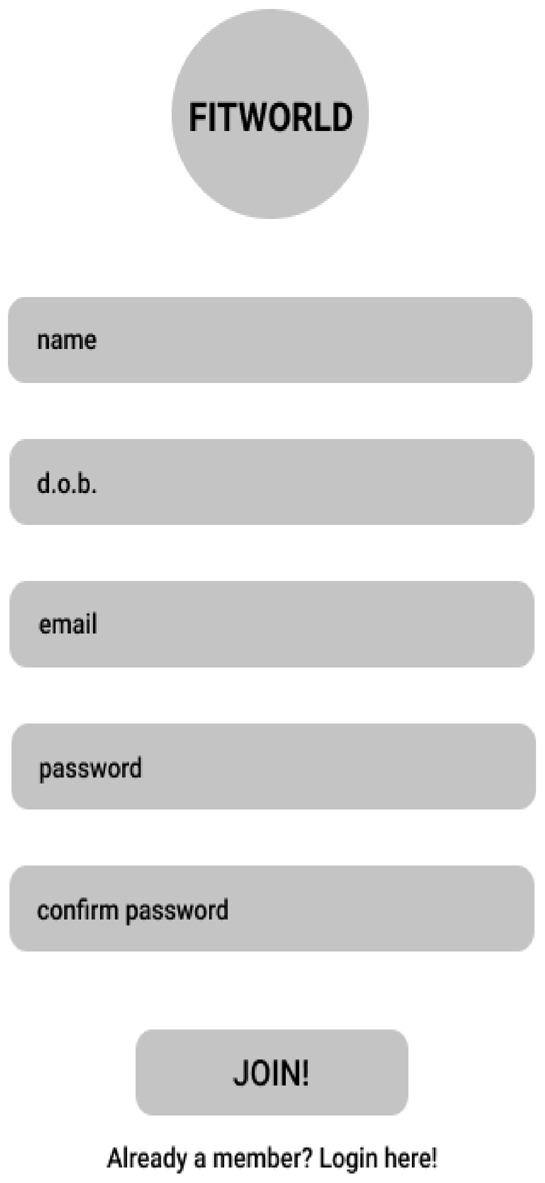
To give users more control over their files and to limit file clutter, apps that target Android 10 (API level 29) and higher are given scoped access into external storage, or *scoped storage*, by default. Such apps have access only to the app-specific directory on external storage, as well as specific types of media that the app has created.

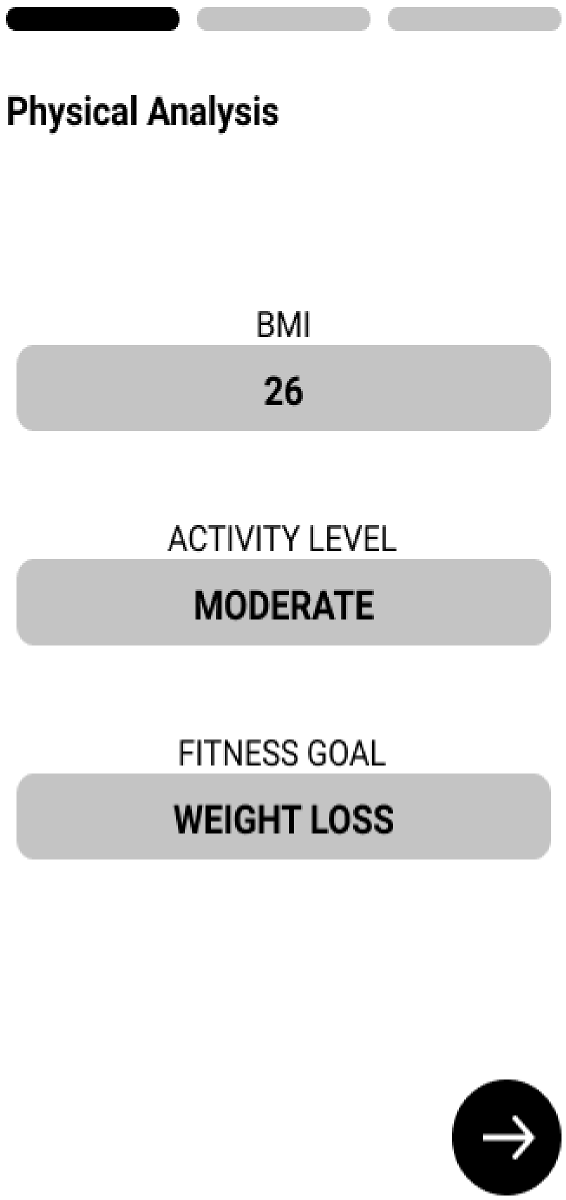
Use scoped storage unless your app needs access to a file that's stored outside of an app-specific directory and outside of a directory that the MediaStore APIs can access. If you store app-specific files on external storage, you can make it easier to adopt scoped storage by placing these files in an app-specific directory on external storage. That way, your app maintains access to these files when scoped storage is enabled.

To prepare your app for scoped storage, view the storage use cases and best practices guide. If your app has another use case that isn't covered by scoped storage, file a feature request. You can temporarily opt-out of using scoped storage.

Result and Evaluation







Physical Analysis

BMI

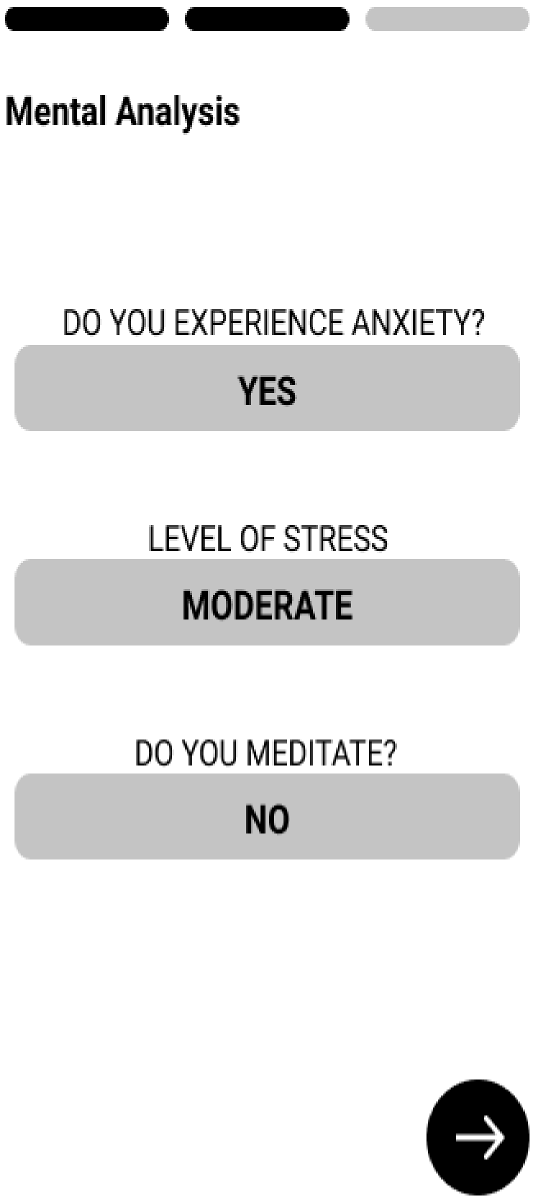
26

ACTIVITY LEVEL

MODERATE

FITNESS GOAL

WEIGHT LOSS



Mental Analysis

DO YOU EXPERIENCE ANXIETY?

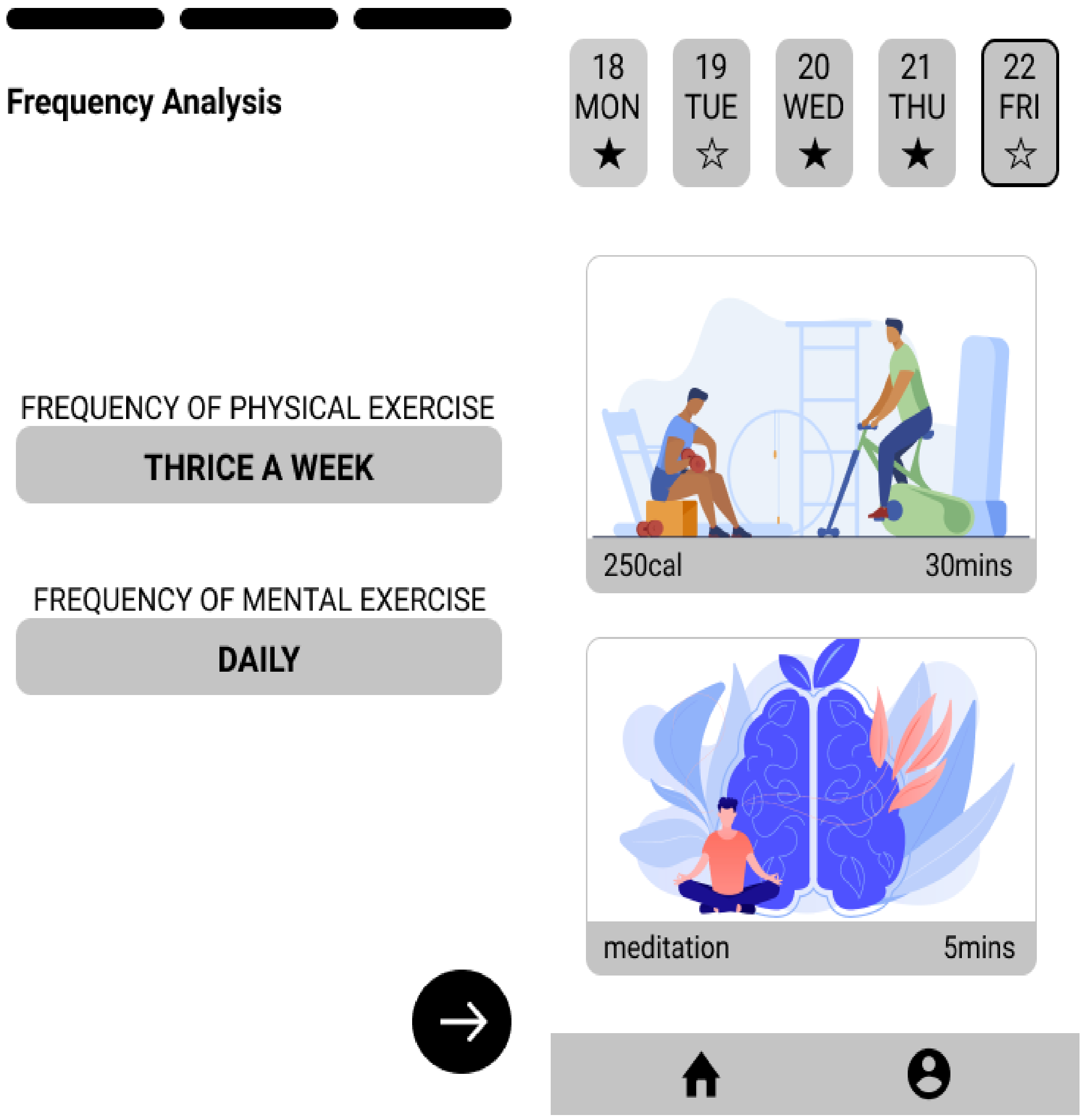
YES

LEVEL OF STRESS

MODERATE

DO YOU MEDITATE?

NO



Frequency Analysis

18 MON ★ | 19 TUE ☆ | 20 WED ★ | 21 THU ★ | 22 FRI ☆

FREQUENCY OF PHYSICAL EXERCISE
THRICE A WEEK

FREQUENCY OF MENTAL EXERCISE
DAILY

250cal 30mins

meditation 5mins

1. Login Page
    a. This page is the authentication page which is used to authenticate the user using their email id and password.
    b. Only registered users are allowed to go past this screen.
    c. If the user is not registered, they are suggested to go to the sign up page.
2. SignUp Page
    a. This page is used to register the user into the application database.
    b. Only registered users are granted access to the application. So every user must register.
3. Physical Analysis
    a. The user is asked a few questions in the page on the basis of which

physical exercises would be recommended to the user in the home page.

4. Mental Analysis
    a. The user is asked a few more questions to assess their mental health situation. On the basis of the answers, mental exercises are suggested to the user.
5. Frequency Analysis
    a. The user fills in the number of times they want to perform these exercises in a week.
    b. Specific days are allotted on the basis of their responses on which exercises are shown on the home screen.
6. Home Screen
    a. Both physical and mental exercises are recommended to the user on this screen.
    b. These exercises are carefully recommended by analysing the responses filled by the user during registration.

Conclusion

We have successfully made our application FITWORLD in which we have provided various features like physical fitness and mental fitness in which we analyze the user database and according to results recommend various exercises.The app also keeps track of the streak maintained and provides a schedule according to their frequency.

The application serves as the first step towards eradicating the problem of obesity and mental trauma that has become a household problem after COVID 19.

Future Aspects

In future, there are a lot of features that can be implemented in the future like:

- Calorie counter
- Step tracker
- Water tracker
- Insightful articles.

All these features would make the application robust and provide a lot of value to the users.

Acknowledgement

We'd like to thank our professors and Galgotias University from the bottom of our hearts for giving us the excellent opportunity to work on this fantastic project FitWorld: Your Fitness Buddy, which has allowed us to conduct significant study and learn about many new things. Second, we'd like to thank Ms. Aanchal Vij, our guide, for her essential support in finishing this project in such a short amount of time.

References

1. https://developer.android.com/docs
2. https://developer.android.com/reference
3. https://firebase.google.com/products/auth?gclid=Cj0KCQiAqbyNBhC2ARIsALDwAsBMNl5rFZn5jBgCe-o1R1AzDe1aTvgZ1LskcsQ5718C8Yjn0iFNJ0IaAuPFEALw_wcB&gclsrc=aw.ds
4. https://firebase.google.com/products/firestore?gclid=Cj0KCQiAqbyNBhC2ARIsALDwAsB5AqHajIjcHWYMo2HdzsCYHfYZYgSzZewTaHcxRom_fhWW-JVbmb8aAiLHEALw_wcB&gclsrc=aw.ds
5. https://www.youtube.com/watch?v=Yt8XkYIdhVU
6. https://www.youtube.com/watch?v=3q3FV65ZrUs
7. https://www.figma.com/resources/learn-design/
8. https://app.diagrams.net/
9. https://kotlinlang.org/docs/android-overview.html